**IN THE CLAIMS**

Sub C1

1. A programming implementation which provides support for reusable software units built in a hierarchical and modular form, comprising: a repository of said reusable software units, where each of said reusable software units are arranged to behave independently, communicate transparently, and facilitate creation of new reusable software units.

2. The programming implementation of claim 1, wherein said reusable software units are arranged to display a scalable mechanism to thoroughly isolate each said reusable unit from its peers.

3. The programming implementation of claim 1, wherein said reusable software units are arranged to facilitate reuse of an overall software system design constructed with said reusable software units.

4. The programming implementation of claim 1, wherein said reusable software units are arranged communicate to the outside world only through the use of input and/or output gates.

5. The programming implementation of claim 4, wherein said reusable software units are arranged to further include a variable mechanism, that abstracts (represents) all the outside software units and whose meaning can only be known at run-time.

6. The programming implementation of claim 5, wherein said reusable software units are arranged in an interconnection of channels (links).

7. The programming implementation of claim 1, wherein said reusable software units do not include absolute addresses of other software units not belonging to its state.

8. The programming implementation of claim 1, wherein said reusable software units do not have the knowledge of the interface (methods) of other software units not belonging to its state.

9. The programming implementation of claim 1, wherein said reusable software units are dynamically configurable.

10. The programming implementation of claim 1, where each reusable software unit may be described by a model $M$, given by

$$M = (inGates, \{inSign_g\}, \{a_g\}, Q, q_0, \text{outGates}, \{outSign_{gt}\}, \{outFunction_{gt}\})$$

where *inGates* is the set of reusable software unit input gates, *outGates* is the set of reusable software unit output gates, $a_g$ is an action for every $g$ in *inGates*, $Q$ is the set of reusable software unit states, $q_0$ is the reusable software unit's initial state, $inSign_{gt}$ is the gate $gt$ input-output signature for each $gt$ in *outGates*, and $outfunction_{gt}$ is the gate $gt$ output function for every $gt$ in *outGates*, where a signature is a 2-tuple containing the range set of incoming and outgoing parameters.

11. The programming implementation of claim 1, wherein a plurality of said reusable software units may be combined to construct a reusable software unit ensemble $E$, defined by

$$E = (inGates, \{inSign_g\}, \varepsilon, M_\varepsilon, outGates, \{outSign_{gt}\}, \{outFunction_{gt}\})$$

where $\varepsilon$ is the ensemble (network) executive that keeps the ensemble structure, and $M_\varepsilon$ is the model of the executive.

12. The programming implementation of claim 11, where the model of the ensemble executive is defined as a model of a reusable software unit augmented with a structure function $\sigma$: $Q \rightarrow \Sigma^*$, where $\Sigma$ is equal to $(C, \{M_c\}, L, \Xi)$; and $C$ is the set of reusable software units that belong to the ensemble, $M_c$ is the definition of each reusable software unit $c$, belonging to set $C$, $L$ is a set of channels, and $\Xi$ is the order function.

13. The programming implementation of claim 11, wherein a channel is a 3-tuple defined by

$$((i, g_i), (j, g_j), (dF, rF))$$

where $i$ is the name of the source reusable software unit, $g_i$ is a gate of $i$, j is the receiver reusable software unit, $g_j$ is a gate of $j$, $dF$ is the channel direct filter and $rF$ is the channel reverse filter.

14. The programming implementation of claims 10 or 11, wherein said reusable software units can be located in: different threads, different processes, different processors and/or different computers.

15. The programming implementation of claims 10 or 11, wherein several of said reusable software units are concurrently active and may access to shared memory.

16. The programming implementation of claims 10 or 11, wherein operations are performed synchronously and/or asynchronously.

17. The programming implementation of claims 10 or 11, wherein said reusable software units can be stored in persistent storage mechanisms.

18. The programming implementation of claims 10 or 11, wherein said reusable software units can be serialized for storing and/or communication purposes.

19. The programming implementation of claims 10 or 11, wherein structural inheritance is utilized to build new reusable software units from existing reusable software units.

20. The programming implementation of claims 10 or 11, wherein objects may be utilized as reusable software units having no output gates.

21. The programming implementation of claims 10 or 11, further including a message breaking mechanism that provides a realization of the variable mechanism according to claim 5.

22. The programming implementation of claims 10 or 11, further including a message breaking mechanism that supports the same effect of the variable mechanism according to claim 5.

23. The programming implementation of claims 10 or 11, wherein a definition of reusable software units and ensembles of reusable

software units are implemented with a graphical tool.

24. The programming implementation of claims 23, wherein said graphical tool is a case tool.

25. The programming implementation of claims 10 or 11, wherein storage can be implemented utilizing one of: a hard disk, a CDROM, a DVD, a floppy disk, and a magnetic tape.

26. The programming implementation of claims 10 or 11, that can be run from either RAM, PROM, EPROM, and EEPROM. Where said implementation may be operated by one of.

Add A29